ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Portál ITStudentHelp - back-end
Student:	Štefan Töltési
Vedoucí:	doc. RNDr. Josef Kolář, CSc.
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2017/18

Pokyny pro vypracování

Tématem práce je analýza, návrh a implementace back-end části webové aplikace (portálu) v rámci širšího zadání, jehož úkolem je vytvořit portál, který bude sloužit jako místo, na které se budou obracet žadatelé o drobné služby související především s osobním využíváním výpočetní techniky, tabletů, mobilních telefonů apod. Vzorové řešení takového portálu představuje https://www.studentaanhuis.nl/.
1. Proveďte analýzu a návrh back-end části. Jedná se zejména o poskytování informací a zpracování požadavků během tlf hovoru s klientem, reakcí na e-mail klienta, a administrování a monitorování portálu.
2. Ve spolupráci s tvůrci ostatních částí portálu proveďte výběr vhodné implementační platformy včetně komunikačního rozhraní mezi back- a front-end částmi portálu. Při návrhu implementační platformy berte v potaz možnost škálování při budoucím nárůstu uživatelů.
3. Navržený back-end implementujte, dokumentujte a otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 2. března 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Portál ITStudentHelp - back-end

Štefan Töltési

Vedoucí práce: doc. RNDr. Josef Kolář, CSc.

9. ledna 2017

Poděkování

Rád bych poděkoval vedoucímu bakalářské práce doc. RNDr. Josefu Kolářovi, CSc. za ochotu uspořádat schůzku, kdykoliv jsem požádal. Dále děkuji rodině za podporu a Fakultě informačních technologií za načerpané znalosti, které jsem při tvorbě bakalářské práce použil.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 9. ledna 2017 .....

Odkaz na tuto práci

Töltési, Štefan. *Portál ITStudentHelp - back-end.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato práce se zabývá analýzou, návrhem, implementací a testováním webového portálu ITStudentHelp. Cílem práce je seznámit čtenáře s postupy použitými ve vývoji zmíněného systému. Portál zákazníkům poskytuje možnost, jak vyřešit problém s výpočetní technikou. Řešení poskytují studenti IT oborů. Hlavní důraz práce je kladen na administrátorskou správu systému a na zpracování nové zakázky přes telefonický hovor.

Jedno z kriterií systému byla škálovatelnost. Proto jsme se rozhodli zvolit technologii Node.js, která se pro naše řešení hodí.

Vytvořené řešení poskytuje plnou administrátorskou správu webového portálu a podporuje veškeré aktivity nutné pro člověka, který poskytuje podporu na telefonu.

Klíčová slova webový portál, ITStudentHelp, backend, návrh, implementace, student IT, pomoc studentů, admin modul, zpracování požadavků telefonního hovoru, škálovatelnost, Node.js, PostgreSQL, Javascript

Abstract

This bachelor thesis is showing the process of analysis, design, implementation and testing of a web portal ITStudentHelp. Its main goal is to familiarize the

reader with methods that have been used in the process of creating the final system. Portal is providing a way how customers can solve their issues with information technology. Solution is provided by IT students. Main emphasis is set on administration and on the process of creating a new task in the system via telephone call.

Scalability was one of the criteria. That is the reason why we have decided to use Node.js. This technology is suitable for our solution.

Created solution is providing full administration of web portal and it is supporting all the activities needed for person who will be providing support on the telephone.

Keywords web portal, ITStudentHelp, backend, design, implementation, IT student, student help, admin module, processing telephone call, scalability, Node.js, PostgreSQL, JavaScript

Obsah

Seznam obrázků

Úvod

V dnešní době moderních technologií stále existují lidé, již se cítí ztraceni v užívání výpočetní techniky a nejsou schopni moderní zařízení plně spravovat. Proto jsem se rozhodl podílet na implementaci webového portálu, který by zmíněným osobám poskytl příležitost, jak vyřešit problémy s ovládáním moderních zařízení. Tito zákazníci budou zadávat zakázky přes frontendovou část systému.

Řešení zakázek budou zajišťovat převážně studenti informačních technologií. Ti po rezervaci úkolu v systému přijdou za zákazníkem a za finanční odměnu mu pomohou s řešením jeho problému.

Služba bude výhodná jak pro zákazníky, tak pro samotné studenty (dále také označováni jako specialisté), jelikož studentům bude poskytovat možnost přivýdělku při studiu a zákazníkům vyřešení potíží za nižší cenu, než jakou by museli uhradit v servisech.

Pro zákazníky, kteří nejsou schopni zadat zakázku do systému, bude existovat člověk na telefonu (dále uváděn jako telefonista), který jejich zadání do systému vloží. Takovýto zaměstnanec bude mít k dispozici webové rozhraní, přes které bude mít možnost plnit požadavky vznikající telefonním hovorem se zákazníkem.

Hlavním zaměřením celého projektu je tvorba dobrého jména Českého vysokého učení v Praze prostřednictvím schopných studentů, kteří na univerzitě studují.

Na úvod je důležité uvést, že mé zadání bylo vytvořeno v rámci širšího projektu, jenž si klade za cíl uvést uvedenou službu do chodu. Konečná systémová rozhodnutí byla tedy konsensem studentů FIT ČVUT pracujících na systému a zadavatelem bakalářské práce.

Studenti, se kterými jsem na projektu spolupracoval, jsou Jan Kabela a Igor Kuľka.

Jan Kabela se zabýval tvorbou databáze a vytvořením části serverového programu, která bude posílat a přijímat požadovaná data frontendové části

portálu. Komunikace s frontendovou částí probíhá ve formátu JSON a prostřednictvím protokolu HTTPS.

Samotnou frontendovou část bežící u klienta měl na starost Igor Kuľka. Část poskytuje grafické rozhraní pro studenty a zákazníky, kde se studenti přiřazují k zakázkám a kde zákazníci zakázky zadávají. Rozhraní po získání informací od klienta posílá data do již zmíněné serverové části systému.

Náplní mé práce bylo vytvoření rozhraní pro administrátora a telefonistu a zároveň vytvoření části serverové aplikace, která bude dané rozhraní obsluhovat. Zabývám se tedy analýzou, návrhem, implementací a testováním části webového portálu podporující funkce vyžadované telefonistou a administrátorem (dále v textu také označováni jako internisté).

Úkolem nad rámec zadání bylo spojení všech částí systému, jelikož zmínění dva spolutvůrci nesloučili své části do jednoho fungujícího celku.

V první části bakalářské práce si stanovím cíle, jež by měl výsledný systém splnit.

Dále se zabývám analýzou a návrhem požadovaného systému. V kapitole 2 definuji funkční a nefunkční požadavky a zároveň popisuji procesy, které jsou od systému vyžadovány. Uvádím zde také diagram databáze, kterou systém využívá.

Následně se v kapitole 3 zabývám popisem použitých technologií v realizační části práce. Snažím se odůvodnit výhody konkrétních technologií a uvádím důvody, proč byly zvoleny. Poté podrobně popisuji, jak některé požadavky byly splněny. V této kapitole rovněž popisuji strukturu konečného projektu a serverové cesty, které část pro internisty nabízí.

Kapitola 1

Cíl práce

Cílem práce je analýza, návrh a implementace backend části webové aplikace (portálu) v rámci širšího zadání, jehož úkolem je vytvořit portál, který bude sloužit jako místo, na které se budou obracet žadatelé o drobné služby související především s osobním využíváním výpočetní techniky, tabletů, mobilních telefonů apod. Vzorové řešení takového portálu představuje https://www.studentaanhuis.nl/.

1. Provést analýzu a návrh backend části. Jedná se zejména o poskytování informací a zpracování požadavků během tlf hovoru s klientem, reakcí na e-mail klienta, a administrování a monitorování portálu.

2. Ve spolupráci s tvůrci ostatních částí portálu provést výběr vhodné implementační platformy. Při návrhu implementační platformy berte v potaz možnost škálování při budoucím nárůstu uživatelů.

3. Integrovat vytvořené části ostatními tvůrci portálu do jednoho kompaktního celku.

4. Navržený backend implementovat, zdokumentovat a otestovat.

Kapitola 2

Analýza a návrh

Před samotnou realizací softwarového produktu je nutno zabývat se analýzou problému, jenž se snažíme vyřešit. V průběhu celého procesu tvorby je důležitá komunikace se zadavatelem, jelikož se snažíme vyhovět jeho požadavkům a stanovit konečné cíle projektu. Komunikace probíhala formou pravidelných schůzek se zadavatelem a ostatními tvůrci systému, aby bylo vyhověno všem systémovým požadavkům.

Požadavky a cíle si specifikujeme před samotnou implementací systému, avšak na úplný začátek se podíváme na již existující a fungující webové portály poskytující servisní služby prostřednictvím studentů vysokých škol.

V další části uvedeme funkční a nefunkční požadavky, jež budou kladeny na část systému implementovanou mou osobou.

Následně představím Use Case diagram, který jsem vytvořil pro celý systém (tedy včetně sekcí pro zákazníka a specialistu).

2.1 Existující řešení

Při hledání již existujících řešení jsem vycházel zejména ze zadání bakalářské práce a ze znalostí zadavatele doc. RNDr. Josefa Koláře, CSc., jelikož trh se systémy podobného ražení sleduje již delší dobu.

Fungujících webových portálů se stejnou funkcí je v zahraničí hned několik, avšak žádný z nich nebyl dostupný v České republice, až do července 2016. V tu dobu byla spuštěna služba Ajťák po ruce, jež se stala přímou konkurencí našeho projektu.

2.1.1 Studentaanhuis.nl

Jedním z existujících řešení je nizozemský portál `https://www.studentaanhuis.nl`. Portál je uveden v zadání bakalářské práce jako referenční projekt, ze kterého čerpáme inspiraci. Studentaanhuis.nl vstoupil do pilotní fáze již v roce 2010, jak je uvedeno v článku [1]. Projekt je v Nizozemsku již zavedený, to dokládá

i ocenění „Service company of the year 2016“ od organizace ICTWaarborg. Koncept je jasný: studenti chodí do domácností a pomáhají zákazníkům se správou zařízení. Řešení problémů je rozděleno do čtyř hlavních kategorií.

- Počítače
- Internet a e-mail
- Mobilní zařízení
- Další zařízení

Pro každou z uvedených možností je možné objednat studenta i pouze na lekce, jak určitá zařízení obsluhovat.

2.1.2 Student@Home

Dalším příkladem, kdy studenti pomáhají s řešením technických problémů, je anglická služba Student@Home `http://www.studentathome.co.uk/` Princip je v základu stejný jako u nizozemského řešení. Momentálně podporu poskytuje více než 80 studentů, a to nejen v Londýně, ale také Birminghamu, Oxfordu, či Canterbury. Opět jsou pro bližší specifikaci zakázky použity kategorie. Jako v případě Studentaanhuis.nl se jedná o zvolení konkrétního zařízení, se kterým zákazník potřebuje pomoci.

2.1.3 Ajťák po ruce

Přímým konkurentem na českém trhu se stala služba Ajťák po ruce `https://ajtakporuce.cz/`, která vznikla v červenci 2016 a funguje na stejném principu, jako již uvedená zahraniční řešení. K řešení problémů se může přihlásit kdokoli, tím pádem zde neexistuje záruka, že se k zákazníkovi dostaví dostatečně znalý specialista.

2.2 Analýza požadavků

Analýza požadavků je důležitou fází ve vývoji softwaru. Při definování požadavků je nutné komunikovat se zadavatelem, aby výsledný produkt odpovídal jeho představám. Právě požadavky definují, jaké funkce lze od výsledného produktu očekávat.

Dle literatury [2] lze požadavek definovat jako specifikaci toho, co by mělo být implementováno. Rozlišujeme dva typy požadavků. Těmito typy jsou:

- Funkční požadavky
- Nefunkční požadavky

2.2.1 Funkční požadavky

Funkční požadavky specifikují, jaké chování lze od koncového systému očekávat, tedy jaké funkce bude plnit. Interní část vytvořená mou osobou má podporovat požadavky vycházející z existence telefonisty a administrátora, proto jsem se rozhodl funkční požadavky rozdělit do dvou skupin.

Požadavky pro telefonistu jsou označené jako FT a pro administrátora jako FA. Následující požadavky se týkají pouze interní části systému.

FT1 Zadání zakázky do systému

Telefonista má k dispozici formulář. Za pomoci formuláře lze zadat novou zakázku do systému. Tento požadavek je specifikován pro případ zákaznického volání (zákazník není schopen z jakéhokoli důvodu zadat zakázku přes webový formulář frontend části systému).

FT2 Zrušení aktivní zakázky

Je možné jakoukoli nedokončenou zakázku zrušit pro případ, že se zákazníkovi podaří vyřešit problém předtím, než ho navštíví specialista.

FT3 Ruční přiřazení zakázky specialistovi

Pokud specialista telefonistovi potvrdí přijetí zakázky, telefonista je schopen přiřadit specialistu k dané validované zakázce.

FT4 Zobrazení seznamu zaměstnanců

Lze zobrazit seznam zaměstnanců evidovaných v systému s jejich kontaktními údaji.

FT5 Zobrazení seznamu zákazníků

Lze zobrazit seznam zákazníků evidovaných v systému s jejich kontaktními údaji.

FT6 Zobrazení seznamu aktivních zakázek

Lze zobrazit seznam zakázek, které doposud nebyly dokončeny.

FT7 Zobrazení seznamu dokončených zakázek

Lze zobrazit seznam zakázek, které byly označeny jako dokončené.

FT8 Úprava zakázky v systému

Lze upravit parametry aktivní zakázky zadané v systému.

FT9 Úprava zákaznických údajů

Telefonista je schopen upravit údaje vedené v systému pro zákazníka.

FT10 Úprava údajů o specialistovi

Telefonista je schopen upravit údaje vedené v systému pro specialistu.

FT11 Vytvoření nového zákazníka

Telefonista je schopen vytvořit nový zákaznický účet.

FT12 Vytvoření nového specialisty

Telefonista je schopen vytvořit nový uživatelský účet pro specialistu.

FT13 Validace zakázek, jež byly zadané zákazníkem

Každou zakázku zadanou do systému zákazníkem musí telefonista zkontrolovat, aby se předešlo vložení nedostatečně specifikované zakázky do systému. Dále telefonista bude moci kontaktovat zákazníka a zjistit další podrobnosti, které bude moci uvést do popisu zakázky.

FA1 Změna role uživatele systému

Administrátor je schopen změnit roli zaměstnance v systému.

FA2 Zobrazení záznamů o fungování portálu

Administrátor má k dispozici textový soubor, kde jsou zaznamenány cesty, na které se uživatel pokouší dostat.

FA3 Úprava často kladených otázek

V zákaznické části se nachází seznam často kladených otázek. Administrátor je schopen seznam upravit.

2.2.2 Nefunkční požadavky

N1 Škálovatelnost konečného systému

Požadavek je uveden již v zadání celé bakalářské práce. Pokud by služba měla být úspěšná, očekává se nárůst uživatelských přístupů na server, proto konečný systém musí být škálovatelný.

N2 Autentizace a autorizace pro cesty dostupné zaměstnancům

Cesty pro administrátora a telefonistu poskytují důvěrné informace o zákaznících a mění data uložené v databázi, proto tyto cesty musí být zabezpečeny proti vniknutí neoprávněné osoby.

2.3 Modelování případů užití

V této části definuji uživatele, již budou konečný systém používat, a uvedu vytvořený Use Case diagram celého systému s důrazem na administrátora a telefonistu.

2.3.1 Uživatelé systému

Pro začátek modelování případů užití je důležité nalézt aktéry. Aktér specifikuje roli, již určitá entita přijímá v okamžiku, kdy začíná daný systém bezprostředně používat. [2]

V následujícím seznamu definuji aktéry, kteří budou konečný systém bezprostředně využívat.

Zákazník

Běžný uživatel, kterému bude poskytována servisní služba. K tomuto uživateli se dostaví specialista a vyřeší jeho konkrétní problém, který byl zadán do systému.

Hlavně se bude jednat o osoby, které nejsou zkušenými uživateli informačních technologií. Jelikož se dá očekávat, že pro takové uživatele bude nepříjemné zadávat zakázku přes webový formulář, bude jim k dispozici zaměstnanec portálu na telefonu. Po dokončení zakázky může zákazník ohodnotit výkon studenta.

Specialista

Zaměstnanec poskytující servisní službu na místě uvedeném zákazníkem. Jedná se o studenty informačních technologií. Každý ze studentů má množinu specializací, která určuje, k jakým zakázkám může být daný student přiřazen. Specializace bude možno měnit.

Telefonista

Zaměstnanec připravený poskytnout přímou podporu zákazníkům a specialistům přes telefon a e-mail. Telefonista je schopen zadávat nové zakázky do systému, má k dispozici seznam všech uživatelů sytému a je schopen validovat zakázky vytvořené zákazníky. V případě nedostatku informací u zakázky

kontaktuje zákazníka a získá více informací potřebných k přiřazení zakázky vyhovujícímu specialistovi.

Administrátor

Zaměstnanec s úplným přístupem k systému. Spravuje role a účty uživatelů v systému. Má přístup ke všem datům uložených v databázi a má na starost správu celého systému.

2.3.2 Use Case diagram

Vytvořený Use Case diagram 2.1 s krátkým popisem všech akcí, jež diagram specifikuje. Jedná se o diagram celého sytému (nejen interní části).

UC1 Aktér specialista zjistí, že v jeho profilu je chybný údaj, či si chce jen upravit popis, který je zobrazen zákazníkům. Údaje upraví ve formuláři v příslušné sekci portálu. Formulář odešle na server, jenž upravená data uloží.

UC2 Aktér specialista vybere ze seznamu zakázku, již může řešit. Následně u zakázky, která mu vyhovuje, potvrdí zájem a zakázka mu bude přiřazena. Od toho okamžiku bude mít kontakt na zákazníka a je možno, aby proběhla dohoda o konkrétním času, kdy se specialista může dostavit na uvedenou adresu.

UC3 Specialistu bude zajímat zpětná vazba od zákazníka, u kterého splnil zakázku. Proto si bude moci zobrazit hodnocení od zákazníka, pokud zákazník hodnocení udělil.

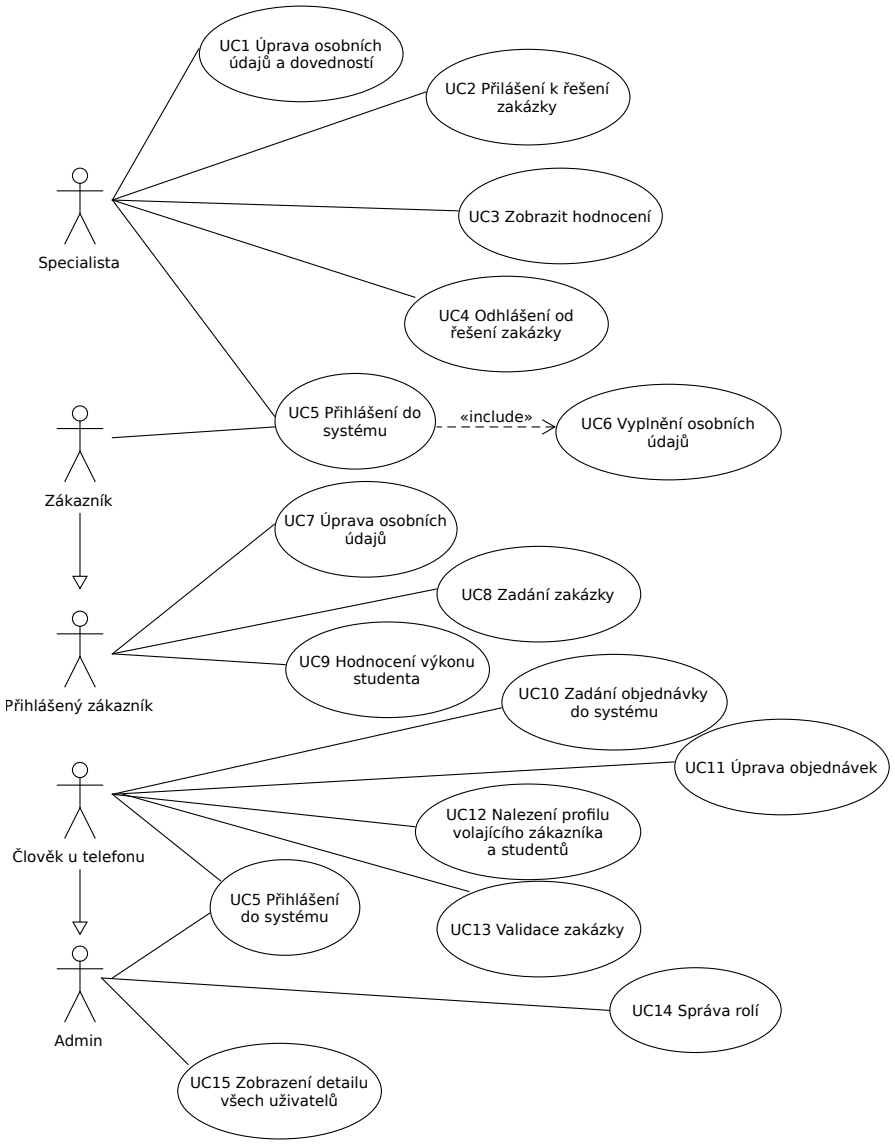
UC4 Specialista nebude schopný domluvit se se zákazníkem na čase, kdy by mohlo být provedeno řešení zakázky. Proto v portálu projeví zájem o zrušení zakázky.

UC5 Uživatel bude chtít využívat funkce systému. Musí se tedy přihlásit a po potvrzení správného e-mailu a hesla může vykonávat akce odpovídající jeho roli v systému.

UC6 Při prvním použití portálu se musí zákazník či specialista zaregistrovat. Musí tedy vyplnit formulář dotazující se na osobní údaje registrujícího se zákazníka.

UC7 Vizte případ UC1. Aktérem je tentokrát zákazník.

UC8 Zákazník má problém k vyřešení, proto se přihlásí a zadá příslušný problém do systému. Ve formuláři se bude snažit co nejpřesněji specifikovat problém, se kterým potřebuje pomoci od specialisty.



Obrázek 2.1: Use Case diagram

UC9 Po vyřešení zakázky specialistou chce zákazník ohodnotit výkon specialisty. Přihlásí se tedy do systému a u vyřešené zakázky zadá hodnocení a komentář k výkonu specialisty.

UC10 Telefonista přijme požadavek od zákazníka pro zadání nové zakázky (přes telefon nebo e-mail). Vyplní tedy příslušný formulář informacemi, jež přijal od zákazníka. Pokud zákazník není v systému, vytvoří se nový účet. Na tento účet bude vedena nová zakázka.

UC11 Telefonista dostane pokyn od zákazníka ke změně informace vedené u zakázky. Může se jednat například o změnu konečné adresy, kde má být servisní služba vykonána. Telefonista může tedy v příslušném formuláři upravit zakázku, případně odhlásit či přihlásit specialistu k řešení zakázky. Úprava bude probíhat pouze u aktivních (dosud neuzavřených) zakázek.

UC12 Aby telefonista mohl poskytovat podporu zákazníkům a specialistům, je nutné, aby byl schopen jednoduše vyhledat profil daného uživatele a následně zobrazit veškeré informace, jež jsou k uživateli v systému vedeny.

UC13 Pokud sám zákazník zadal zakázku do systému, není jisté zdali poskytl dostatečný popis problému, který má specialista vyřešit. Zakázku tedy musí validovat telefonista, který v případě nedostatku údajů kontaktuje zákazníka a doplní či opraví údaje. Po validaci je zakázku možné přiřadit specialistovi k řešení.

UC14 Bude potřeba změnit roli a tedy i práva uživatele. Administrátor nalezne příslušného uživatele a v příslušném formuláři vyplní role, jež má uživatel zastávat.

UC15 Po administrátorovi bude vyžadována úprava údajů telefonisty. Jelikož administrátor má právo ke změně jakýchkoli údajů, upraví údaje v příslušném formuláři.

2.4 Procesní modely

V následující části se zaobírám analýzou procesů vycházejících z již uvedených funkčních požadavků a Use Case diagramu. Pro znázornění průběhu procesu používám Business Process Model notaci (zkráceně BPMN).

BPMN poskytuje organizaci možnost porozumět interním procedurám a dává organizaci možnost zachytit tyto procedury ve standardizované grafické podobě. [3]

Procesní model slouží také jako dokumentace vytvořeného systému. Díky němu je schopen člověk neznalý vnitřního fungování sytému pochopit sousledost akcí, které jsou vykonány pro určitý vstup. Diagramy jsou vytvořeny dle

normy BPMN verze 2.0. Specifikace BPMN lze nalézt v [3] v sekci BPMN v2.0.

2.4.1 Průběh telefonního hovoru

Diagram 2.2 znázorňuje průběh telefonního hovoru mezi telefonistou a zákazníkem, který má zájem o specifickou servisní službu. Pokud je zákazník již evidován v systému, lze ho najít v seznamu zákazníků. K rychlému nalezení pomůže telefonistovi vyhledávání v seznamu již evidovaných zákazníků. Pokud zákazník není evidován, vytvoříme mu nový uživatelský profil s vyplněnými osobními údaji.

2.4.2 Přidělení zakázky

Obrázek 2.3 přibližuje průběh přidělení zakázky specialistovi. Existují dva způsoby, jak se zakázka může dostat do stavu přiřazena.

Specialista si po přihlášení sám zakázku vybere ze seznamu dostupných zakázek. Další možností je, že telefonista kontaktuje specialistu, ten projeví zájem o zakázku, která mu je následně telefonistou přiřazena.

Při přiřazení zakázky telefonistou je odeslán informační e-mail přiřazenému specialistovi.

2.4.3 Validace zakázky

Na 2.4 popisuji proces validace zakázky. Telefonista validuje pouze zakázky, které byly zadány zákazníky. Ze serverové aplikace se pošle seznam nevalidovaných zakázek do rozhraní telefonisty. Telefonista vybere jednu zakázku ze seznamu a zkontroluje údaje uvedené zákazníkem. Při správnosti všech údajů označí zakázku jako validovanou a pošle požadavek o validaci na server, který změní stav zakázky.

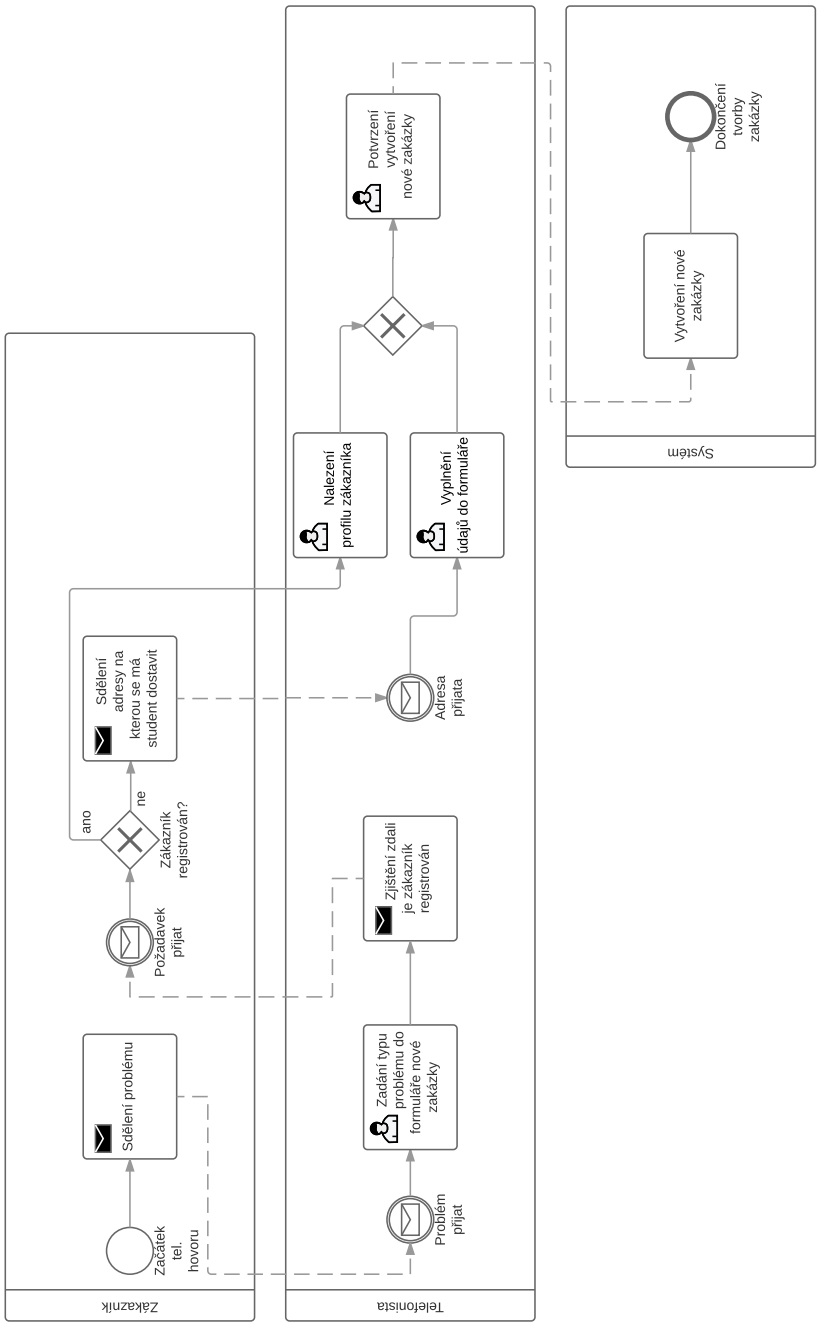
V případě výskytu nepřesného či chybného údaje kontaktuje telefonista zákazníka za účelem zpřesnění nebo opravy detailů uvedených u zakázky. Po opravení detailů zakázky telefonista odešle validovaná data na server.

Pokud zakázku vytvoří telefonista, stává se automaticky validovanou.

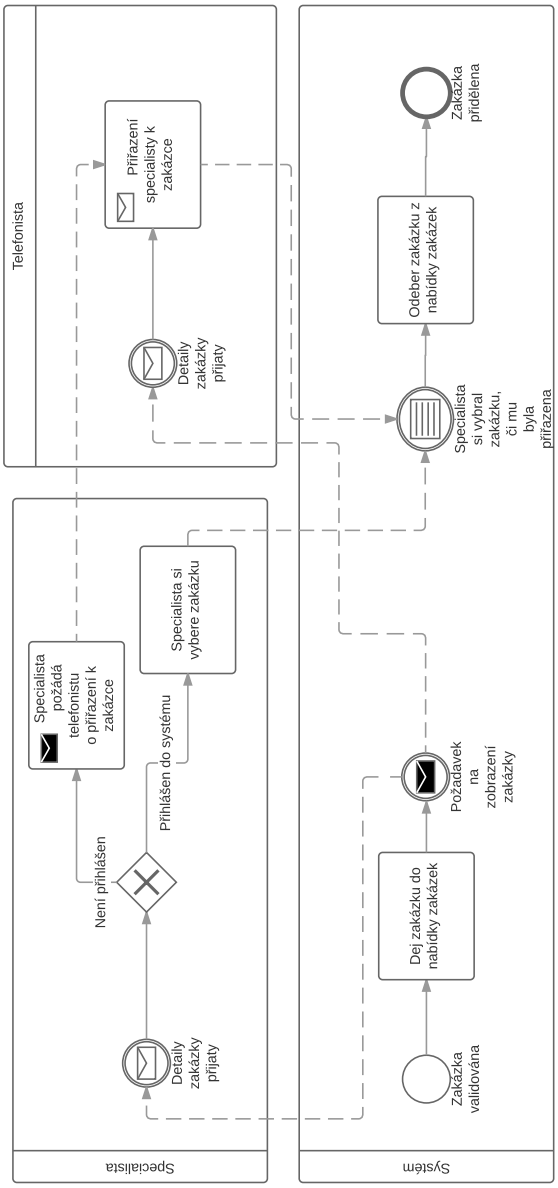
2.4.4 Editace dat

Diagram 2.5 dokumentuje změny údajů uložených v sytému pro zákazníka, zaměstnance či zakázku. Po zobrazení stávajících údajů vedených pro vybranou kategorii změní telefonista chybné údaje v příslušném formuláři. Formulář následně odešle na server, který změněná data uloží do databáze.

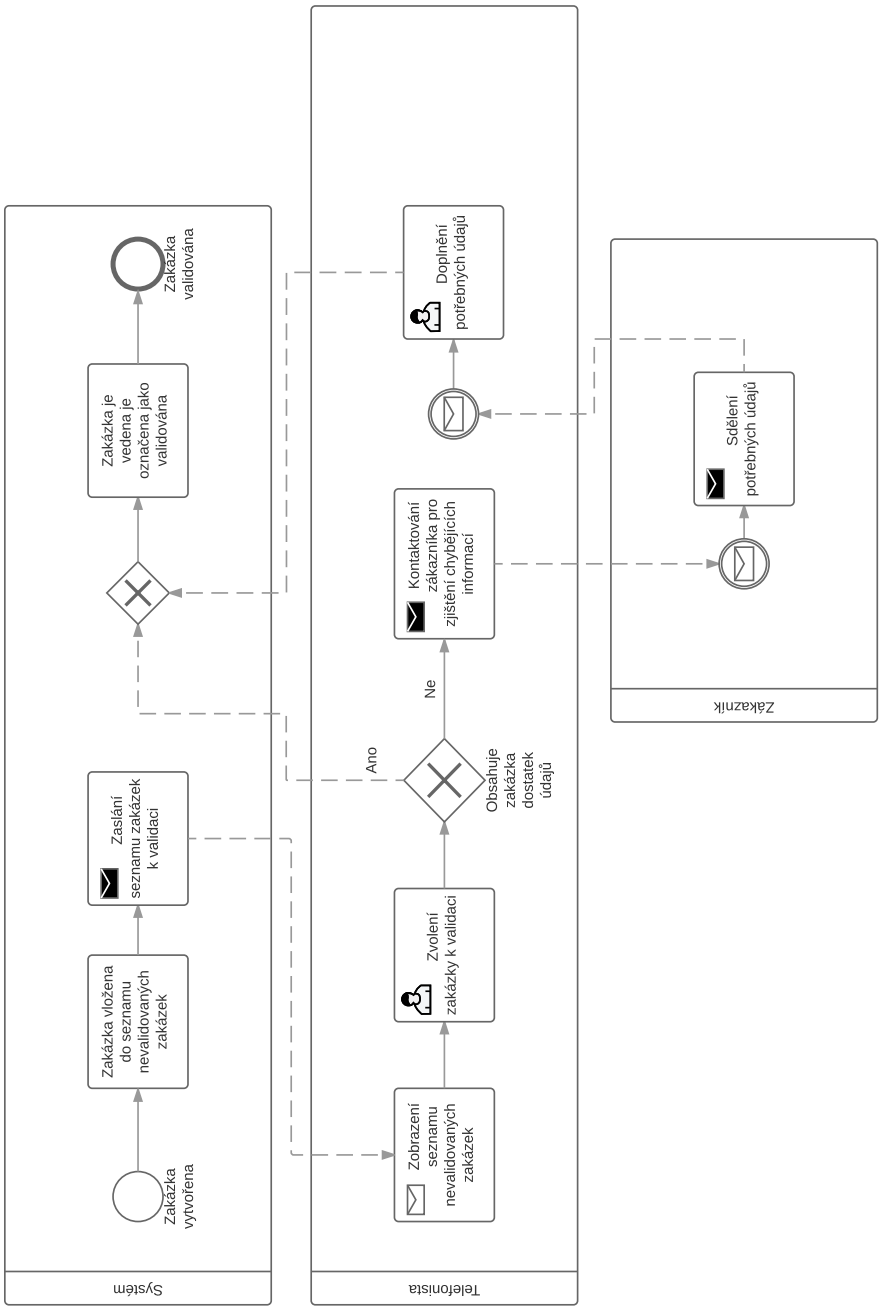
V následující části uvedu, jaké údaje lze konkrétním kategoriím měnit.



Obrázek 2.2: Průběh telefonního hovoru



Obrázek 2.3: Průběh přidělení zakázky specialistovi



Obrázek 2.4: Proces validace zakázky

Zakázka

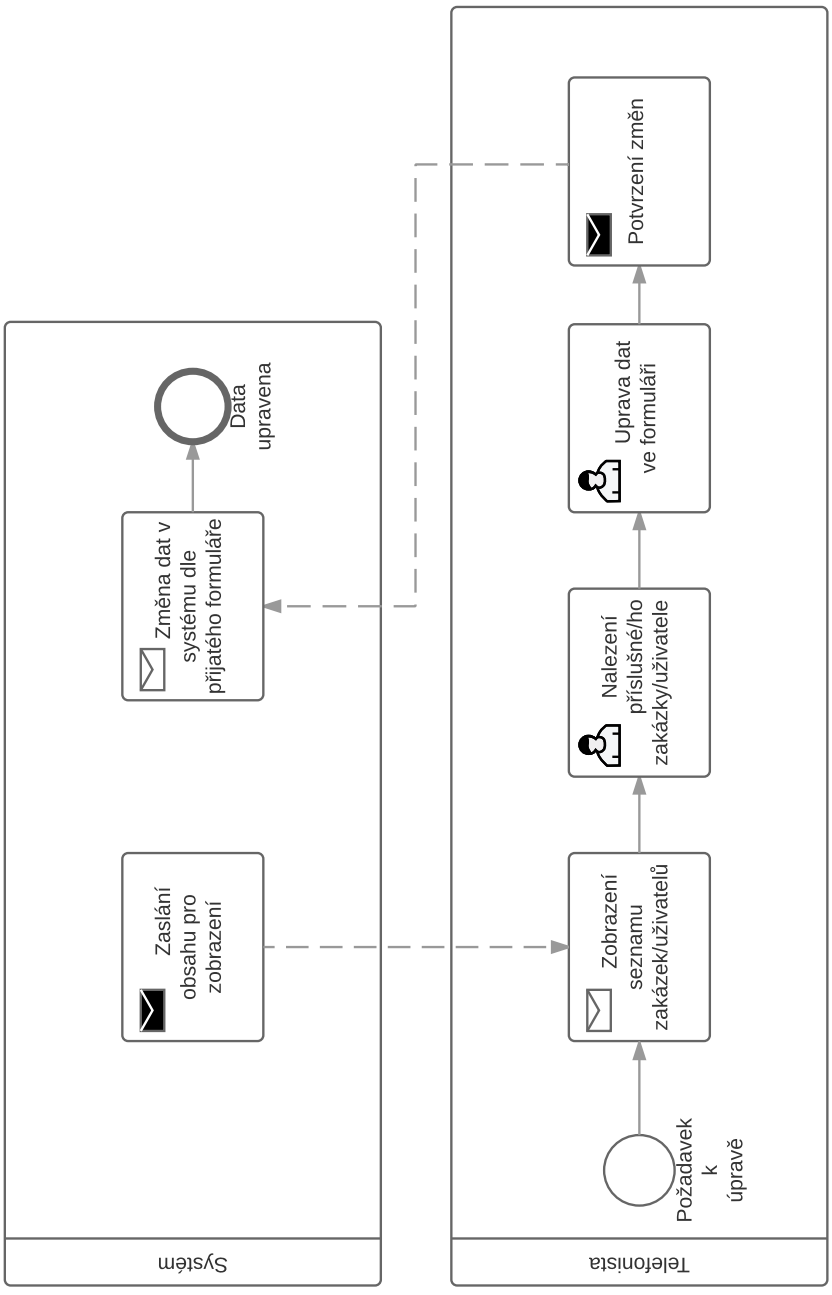
- Popisek: Krátký popis úkolu určený pro výpis v přehledech zakázek.
- Popis: Přesný popis, co je třeba vyřešit u zákazníka. Potřebné pro specialisty, kteří si zakázky budou volit.
- Specializace: Zařazení do kategorie specializací. Zařazení bude sloužit pro výběr správného specialisty k zakázce.
- Adresa: Údaje o adrese, kde má být zakázka vykonána.

Zaměstnanec

- Jméno
- Příjmení
- Datum narození
- Univerzita: Jelikož v systému mohou být studenti nejen z ČVUT, ale i z jiných univerzit.
- Telefon
- Adresa
- Popis: Uvidí zákazníci při zobrazení detailu zaměstnance.
- Specializace: Seznam specializací, kde specialista bude mít zaškrtnuté ty, které ovládá.

Zákazník

- Jméno
- Příjmení
- Adresa
- Telefon



Obrázek 2.5: Proces editace dat (uživatele nebo zakázky)

2.5 Životní cyklus zakázky

Pro porozumění procesů v systému popisuji životní cyklus zakázky od jejího vzniku až po její dokončení. Vše dokumentuje vytvořený stavový diagram 2.6.

Zakázku lze založit dvěma způsoby. Buď ji založí přímo zákazník, který zadá potřebné údaje přes frontend systému, nebo bude vytvořena telefonistou, který přijal požadavek pro vytvoření nové zakázky po telefonu nebo e-mailu. Pokud zakázku vytvořil telefonista, stává se automaticky validovanou.

Při založení zákazníkem zakázka přechází do stavu „vytvořena“ a čeká ve frontě na validaci telefonistou. Ten vybere konkrétní zakázku z fronty a zkontroluje u ní veškeré uvedené údaje. Pokud zákazník nebyl dostatečně konkrétní nebo zadal chybný údaj, kontaktuje telefonista zákazníka za účelem doplnění informací.

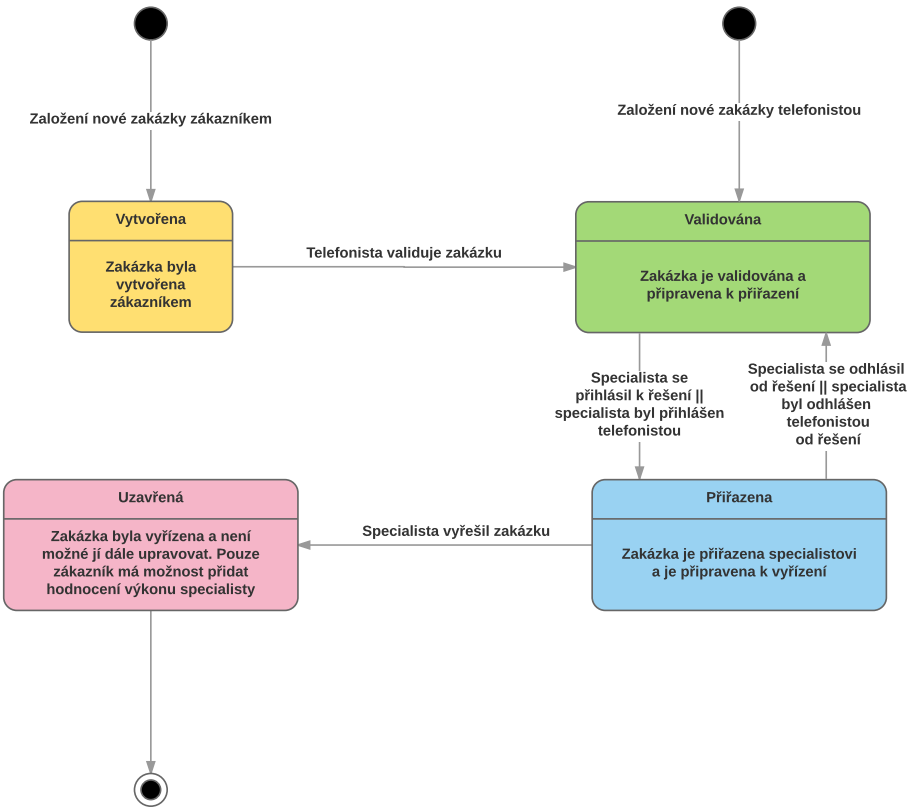
Pokud jsou uvedená data korektní, zakázka přechází do stavu „validována“. Validované zakázky si již mohou prohlížet studenti specialisté. Pokud se jim konkrétní zakázka zalíbí, jsou schopni se přiřadit k jejímu řešení. Proces je detailně popsán v kapitole 2.4.2.

Po přiřazení specialisty se stává zakázka „přiřazenou“. Specialista vykomunikuje čas vykonání zakázky buď přímo se zákazníkem, nebo zašle telefonistovi možné termíny, kdy by se mohl na určenou adresu dostavit. Následně se telefonista domluví se zákazníkem na přijatelném čase.

Pokud z jakéhokoliv důvodu ztratí specialista zájem o zakázku, ke které byl přiřazen, telefonista je schopný ho od zakázky odhlásit.

V případě, že specialistovi čas vyhovuje, dostaví se k zákazníkovi a jeho problém vyřeší. Po úspěšném dokončení úkolu zákazník podepíše papírový formulář o délce provedené práce a předá ho specialistovi. Tomu bude formulář sloužit jako důkaz o strávené době u zákazníka. Potvrzený čas také zapíše do systému k evidenci odpracovaných hodin. Tímto se zakázka dokončí. Pokud specialista nebude moci zadat odpracovanou dobu sám, požádá telefonistu, aby zakázku označil jako dokončenou a zadal strávený čas u zákazníka do systému.

Poté co byla zakázka dokončena, se bude nacházet v interní části pro telefonistu v archivu zakázek. Nyní již nelze se zakázkou manipulovat. Pouze zákazník je schopen z frontendové části ohodnotit výkon specialisty, který zakázku provedl.



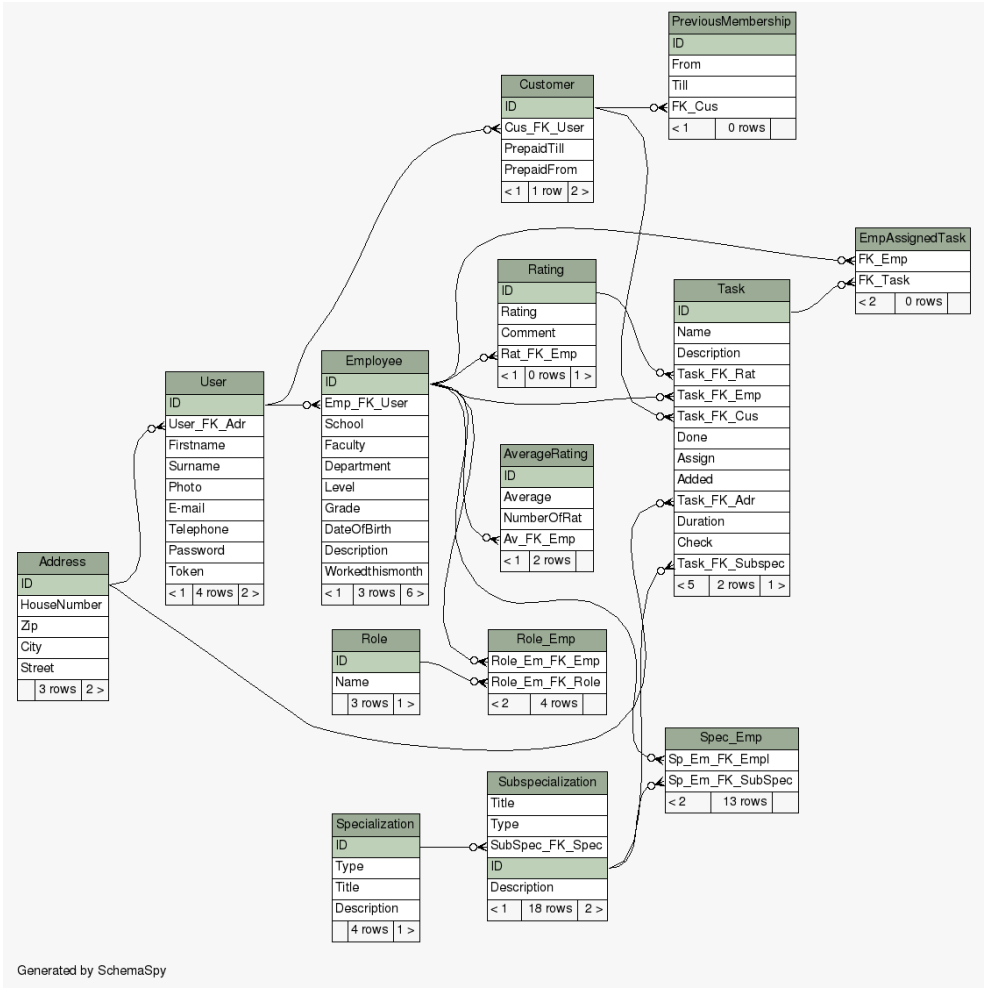
Obrázek 2.6: Stavový diagram životního cyklu zakázky

2.6 Návrh databáze

V této části uvedu schéma databáze 2.7, kterou celý systém využívá. Hlavním tvůrcem návrhu je Jan Kabela, avšak považuji za důležité uvést zde databázové schéma, podle kterého lze poznat, s jakými objekty je nutno počítat v samotné implementaci serverového programu. Schéma je generováno z již vytvořené databáze pomocí nástroje SchemaSpy, jelikož Jan Kabela neudržoval E-R diagram návrhu databáze aktualizovaný. Na návrhu jsem se podílel několika změnami a připomínkami.

Například se jednalo o vytvoření is-a hierarchie pro uživatele (user), zákazníka (customer) a zaměstnance (employee). Společné atributy entit zákazníka a zaměstnance byly tedy přesunuty do entity uživatele. Při transformaci do relačního schématu Jan Kabela zvolil řešení, kde pro každou z entit vytvoříme vlastní tabulku.

Dále bylo v původním návrhu použito datového typu char fixní délky



Obrázek 2.7: Vztahy entit v databázi

255 znaků pro všechny atributy řetězcové povahy (jméno, příjmení, popis zakázky...). Pokud byl řetězec kratší než uvedených 255 znaků, byl doplněn prázdnými znaky do požadované délky. V databázi byl tedy záznam uložen vždy s maximální možnou délkou. Zpravidla však řetězce uložené v databázi mají délku výrazně kratší, proto jsem navrhl změnu datového typu na varchar.

Výhodou tohoto typu je, že řetězec již není vyplněn prázdnými znaky, je tedy podstatně menší a nepotřebuje takové množství místa pro uložení. Krok je na úkor rychlosti vyhledávání v databázi, jelikož práce s řetězci variabilní délky je časově náročnější.

Kapitola 3

Realizace

V této kapitole se věnuji samotné implementaci systému. Je zde popsáno, jak konkrétně jsou některé požadavky splněny a jaké způsoby řešení jsem zvolil.

Také zde uvádím strukturu odevzdaného systému, společně s popisem jednotlivých částí a konkrétních funkčností.

3.1 Nástroje, návrhové vzory a technologie použité při implementaci

V této kapitole se budu zabývat nástroji a technologiemi, které byly použity v průběhu tvorby projektu. Budu se také snažit odůvodnit, proč byly dané nástroje a technologie použity a jaké výhody přinášejí. Pro návrhové vzory uvedu, jak jsou v konečném řešení použity.

3.1.1 MVC architektura

Model View (česky pohled) Controller (česky řadič) je softwarová architektura, běžně užívaná pro implementaci uživatelských rozhraní, z tohoto důvodu se jedná o oblíbenou volbu pro tvorbu webových aplikací. Rozděluje vnitřní logiku aplikací do tří oddělených částí, díky tomu je zajištěna modularita a opětovná použitelnost. Rozdělení na části také dělá aplikaci flexibilní a více přívětivou dalším iteracím. [4]

Model je vrstva starající se o poskytování dat řadiči. Dále od řadiče přijímá požadavky na změnu specifických řádků databáze. V našem případě je vrstva implementována v technologii Node.js a v mém projektu se nachází v adresáři Server/routes/intern/modelFunctions. Model získává data z databáze prostřednictvím SQL dotazů.

View (pohled) je vrstva prezentující získaná data uživateli. Definuje tedy, jak jsou data zobrazena. V implementaci lze pohled pro mou část bakalářské práce nalézt v adresáři Server/views/intern.

Poslední vrstvou je Controller (řadič). Ten funguje jako prostředník mezi modelem a pohledem. Nachází se v něm aplikační logika a upravuje data do tvaru, v jakém jsou očekávána v pohledu.

3.1.2 Node.js

Zde si představíme Node.js. Jak uvádí zdroj [5], Node.js je běhové prostředí postavené na V8 JavaScriptovém jádře. Používá neblokující I/O model, který dělá Node.js lehkým a efektivním řešením.

Zde se dostáváme k nefunkčnímu požadavku N1 škálovatelnosti, který použitím této technologie splníme. Jedná se sice o prostředí běžící pouze na jednom vláknu, avšak právě díky tomu, že je neblokující, je ideální pro servery, které neprovádějí složité výpočty. Jako příklady použitelnosti Node.js pro velké množství přístupů mohou sloužit weby společností jako je PayPal, Amazon, Ebay nebo Yahoo. [6]

Důvodem, proč je možné obsloužit velké množství uživatelů pouze jedním vláknem, je fakt, že aktivity, u kterých čekáme delší časový úsek na dokončení, jsou neblokující. Díky tomu není potřeba čekat, než dostaneme například data z databáze, ale pokračujeme v běhu programu dále a po dokončení časově náročné akce se vrátíme k získaným prostředkům pomocí callback funkce.

Pro tvorbu systému jsem použil Node.js ve verzi 4.7.0. Je důležité interpretovat zdrojové kódy právě v této verzi, jelikož starší verze nepodporují funkčnosti některých knihoven, které jsem při implementaci použil.

3.1.3 PostgreSQL

Jako databázový systém byl zvolen open source objektově-relační PosgresSQL. Systém byl zvolen z důvodu snadného propojení se serverem psaným v Node.js. Dalším důvodem byla skutečnost, že není nutné platit licenční poplatky za využívání systému v případě, že dojde k úspěšnému nasazení.

3.1.4 Jade

Jedná se o modul do Node.js pro tvorbu HTML šablon. V mém případě se tedy jedná o nástroj, který jsem použil pro tvorbu pohledu v klasické MVC architektuře. Umožňuje iterovat přes pole, která mu jsou předána, a zkracuje zápis HTML kódu.

3.1.5 Express

Express je flexibilní, minimalistický framework pro Node.js, který nabízí robustní množinu funkcí určenou pro tvorbu webových a mobilních aplikací. [7]

Jelikož je Express pouze tenkou nástavbou na Node.js, stal se základním stavebním kamenem spoustě dalších frameworků.

Použití umožňuje rychlejší psaní kódu, nežli by bylo možné v samotném Node.js. Express poskytuje jednoduché směrování požadavků, které jsou na server zaslány. Dále nabízí spoustu modulů, které lze v aplikaci použít. V mém případě například používám parser příchozích GET a POST požadavků a modul pro vedení session, užívaný v interní sekci pro autorizaci a autentizaci.

3.1.6 Passport

Passport je middleware Node.js využívaný k autentizaci. Velmi flexibilní a modulární Passport může být zakomponován do jakékoliv aplikace stojící na základech frameworku Express. Využívá obsáhlou množinu strategií podporující autentizaci za pomoci uživatelského jména a hesla, Facebooku, Twitteru a dalších. [8]

Výhodou je jednoduché zakomponování middlewaru do frameworku Express. Dalším pozitivem je snadné rozšíření strategií k přihlašování. V budoucnu tedy bude možné využít například přihlášení přes Facebookový účet apod.

V projektu je Passport využit pro autorizaci a autentizaci v interním modulu. Více o využití lze nalézt v kapitole 3.3.

3.1.7 Datatables

Jedná se o zásuvný modul pro Javascriptovou knihovnu jQuery. Poskytuje pokročilé možnosti HTML tabulkám. [9]

V systému využívám modul pro zobrazení dat v části určené internistům. Tabulky jsou obohaceny o řazení dle určitého sloupce a o vyhledávání výskytu řetězců v dané tabulce. Tímto je zajištěno rychlé vyhledání profilu volajícího zákazníka či specialisty.

3.1.8 GIT

Jedná se o verzovací systém, který je téměř nezbytný pro projekty tvořené skupinou programátorů. Jelikož jsem na systému ITStudentHelp pracoval s dalšími studenty FIT ČVUT, zvolili jsme git pro sdílení zdrojového kódu. Kód byl nahráván na `https://gitlab.fit.cvut.cz`, což je server poskytnutý fakultou pro projekty studentů.

3.1.9 npm

Jedná se o balíčkovací systém, který je základní součástí instalace Node.js. Díky npm jednoduše stáhneme moduly, které náš projekt využívá.

Pro stažení veškerých závislostí lze využít package.json, který je standardní součástí každého Node.js projektu.

3.2 Průběh integrace částí systému

Jelikož spolutvůrci systému Jan Kabela a Igor Kuľka odevzdali bakalářskou práci v letním semestru 2016, bylo mým úkolem integrovat jejich dvě části, které na sobě byly závislé, do jednoho kompaktního celku společně s částí vytvořenou mou osobou. Nutno dodat, že zmínění spolutvůrci nestihli otestovat komunikaci mezi jejich dvěma částmi. Závislost obou prací je následující.

Část frontendu (Igor Kuľka) prezentuje data zákazníkům a specialistům. Data jsou získána ze serverové části systému (vytvořené Janem Kabelou). Proto je patrné, že části musí komunikovat přes definované rozhraní.

Za tímto účelem vytvořil Igor Kuľka návrh API, dle kterého má komunikace probíhat. Návrh byl vytvořen ve webové aplikaci apiary, zde se nachází přesný popis s definicí formátu dat a se specifikací cest, kde se daná data nachází.

Díky navrženému API měla být integrace rychlá, avšak při testování funkčnosti jsem narazil na několik problémů.

Ve verzovacím systému GIT se dlouhou dobu vyskytoval zastaralý kód backendové části. Díky tomu se v souborech nacházelo značné množství chyb, které jsem musel opravit k zajištění správné funkčnosti. Vzhledem k závažnosti chyb jsem se rozhodl znovu kontaktovat autora. Vyšlo najevo, že soubory, které byly odevzdány k bakalářské práci Jana Kabely, nebyly zaneseny v systému GIT (funkcí commit). Po obdržení finálních souborů jsem opět začal s testováním komunikace.

Při procházení systémem jsem narazil na spoustu rozdílností v samotné implementaci backendové části a definovaným API.

Některé cesty, na kterých měla být dostupná data pro frontend, byly špatně uvedeny. Kupříkladu se jednalo o následující (první je uvedená původní cesta, poté opravená dle API).

- api/register - api/register/user
- api/edituser - api/edit/user
- api/activeissues/(id uživatele) - api/profile/(id uživatele)/active-issues
- api/task/addrating - api/review
- api/activetasks/(id úkolu) - api/profile/(id úkolu)/active-tasks
- api/taskpool/(id úkolu) - api/profile/(id úkolu)/task-pool

Po opravě byl frontend schopen získávat data ze serveru ve formě Javascriptového objektu, pomocí jeho textové reprezentace v notaci JSON. Bylo však nutné upravit vlastnosti (properties) objektů, které se předávaly. Přesněji bylo nutno opravit názvy vlastností, které byly zasílány, jelikož se frontend snažil

```
1  var formidable = require('formidable');
2
3  module.exports = function parseFormData(req, res, next) {
4      var contype = req.headers['content-type'];
5      if (contype.indexOf('multipart/form-data') !== -1) {
6          var form = new formidable.IncomingForm();
7          form.parse(req, function (err, fields, files) {
8              req.body = JSON.parse(fields.json);
9              if (files)
10                 req.files = files;
11             next();
12         });
13     } else
14         next();
15 };
```

Obrázek 3.1: Využití modulu formidable

získat data z obdrženého objektu z vlastností nesoucích odlišné jméno. Názvy vlastností objektu, které frontend využívá, jsou definovány ve zmíněném API vytvořeném v apiary.

Dále některé vlastnosti (properties) objektu server vůbec neposílal. Respektive se je zasílat pokoušel, avšak při tvorbě objektů k zaslání na frontend vyplnil backend zasílaný objekt nedefinovanou hodnotou. Často se jednalo o časové záznamy. Například čas přidání úkolu do systému a čas přiřazení úkolu studentovi.

Problémem také bylo, že server očekával formulářová data pro metodu HTTP POST kódována pomocí application/x-www-form-urlencoded. Jak uvádí zdroj [10], pomocí tohoto postupu jsou data z formuláře zakódována přímo v URL. Používáme tedy stejný prostředek, jaký bychom využili u metody HTTP GET. Tento postup se nehodí pro zasílání obrázků a větších souborů, proto frontend zasílá data v kódování multipart/form-data.

Řešením problému bylo využití modulu pro Node.js s názvem Formidable. Dokumentace je dostupná z následujícího zdroje [11]. Ve výše uvedené ukázce kódu 3.1 je konkrétní použití modulu v systému.

Lze vidět, že ukázka 3.1 je využívaná jako middleware. Vyparsovaná data jsou uložena tak, jako kdyby byla zaslána kódováním application/x-www-form-urlencoded, tedy v req.body na řádce 8. Kód je k nalezení v souboru Server/functions/parseFormData.js.

V ukázce 3.2 se nachází použití kódu ze souboru parseFormData.js jako middlewaru pro cesty začínající řetězcem api, na které byla zaslána data metodou POST protokolu HTTP.

Nakonec se mi podařilo integrovat obě části, aby fungovaly ve vzájemné součinnosti a poskytovaly základní funkčnosti zákazníkům a specialistům. Nyní komunikace probíhá dle definovaného api, to lze nalézt na následující

```
1 var parseFormData = require('./functions/parseFormData');
2 app.post('/api/*', parseFormData);
```

Obrázek 3.2: Užití kódu parseFormData jako middlewaru

adrese `http://docs.itstudenthelp.apiary.io/#` nebo v příloze bakalářské práce.

3.3 Zabezpečení cest přístupných internistům

Jak je uvedeno v nefunkčním požadavku N2, cesty pro internisty poskytované serverem musí být chráněné proti zneužití neoprávněnými osobami. Požadavek jsem splnil díky použití middlewaru Passport.

Pro přihlašování do interního modulu jsem zvolil strategii s názvem local. Strategie využívá uživatelské jméno (v mém případě e-mail) a heslo. Pomocí této dvojice je uživatel schopen se přihlásit do interní části systému.

Při zadání přihlašovacích údajů získá klientská strana od serveru unikátní řetězec znaků, pomocí kterého se autentizuje v navazujících žádostech. Řetězec je předán v hlavičce HTTP žádosti a uložen u klienta. Jedná se tedy o využití cookies.

V ukázce 3.3 lze nalézt kód definice strategie, kterou pro interní část využívám. Na řádce číslo 8 zjišťuji, zdali příchozí požadavek o přihlášení obsahuje e-mail, který je v databázi přidělen internistovi.

Pokud ano, pak kontrolujeme na řádce 17, zdali je příchozí heslo korektní pro nalezeného uživatele. V případě, že vše proběhlo v pořádku, posíláme údaje o zákazníkovi do callback funkce done. Zde dochází k serializaci uživatele pro kontrolu správnosti cookie, kterou budeme dostávat od klientské strany v budoucích dotazech.

Samotná kontrola, zdali je uživatel správně přihlášen, se provádí při každém pokusu o přístup do interní části. Kontrola se provádí jednoduše díky použitému middlewaru Passport. To demonstruji v ukázce 3.4. Pokud uživatel není přihlášen, automaticky ho přesměrujeme na přihlašovací obrazovku.

Další nutností bylo rozlišení, zdali je přihlášen telefonista nebo administrátor, z důvodu existence cest, které mají být přístupné pouze administrátorovi. Mezi ně patří následující:

- intern/employee/changeRole - cesta sloužící ke změně rolí uživatelů
- intern/faq - cesta, pomocí které administrátor přidává a odebírá otázky/odpovědi pro sekci často kladené otázky ve frontendové části

```
new LocalStrategy({
      usernameField: 'username',
      passwordField: 'password',
      passReqToCallback: true
  },
          function (req, username, password, done) {
              username = username.toLowerCase();
              db.any('SELECT "User"."ID","Password","Role"."
                  Name" as role FROM' +
                       '"User" JOIN "Employee" ON "User"."ID"="
                          Emp_FK_User"' +
                       'JOIN "Role_Emp" ON "Employee"."ID" = "
                          Role_Em_FK_Emp"' +
                       'JOIN "Role" ON "Role"."ID"="
                          Role_Em_FK_Role"' +
                       'WHERE "E-mail"= $1 AND "Role"."Name"=\'
                          Admin\' OR "Role"."Name"=\'
                          Telephonist\'', [username])
                       .then(function (result) {
                           if (!result.length) {
                               return done(null, false, req.
                                   flash('loginMessage', 'Wrong
                                   e-mail.'));
                           }
                           if (!isValidPassword(result[0],
                               password)) {
                               return done(null, false, req.
                                   flash('loginMessage', 'Wrong
                                   password.'));
                           }
                           return done(null, result[0]);
                       }).catch(function (err) {
                   return done(err);
              });
          }
  )
```

Obrázek 3.3: Definice strategie pro Passport

```
function isLoggedIn(req, res, next) {
  if (req.path.indexOf('/intern/login') !== -1 )
      return next();
  if (req.isAuthenticated())
      return next();
  res.redirect('/intern/login');
};
```

Obrázek 3.4: Kontrola zdali je uživatel přihlášen

```
module.exports = function (req, res, next) {
  if (req.user) {
      if (req.user.role !== 'Admin') {
          var routes = [
              'intern/employee/changeRole',
              'intern/faq',
              'intern/logs'
          ];
          for (var i = 0; i < routes.length; i++) {
              if (req.path.indexOf(routes[i]) !== -1)
                  return res.render('intern/error/forbidden');
          }
      }
  }
  next();
};
```

Obrázek 3.5: Middleware zajišťující ochranu administrátorských cest

- intern/logs - cesta, díky které lze prohlížet logy systému

Řešení omezeného přístupu k administrátorským cestám lze nalézt na 3.5. Nejdříve zjistíme, jestli je přihlášený uživatel administrátorem.

Pokud ano, pak není omezen přístup na cestu příchozího požadavku. Pokud uživatel nemá v systému roli administrátora, kontrolujeme, zdali se pokouší dostat na jednu z cest s omezeným přístupem. V případě, že se jedná o cestu, ke které nemá oprávnění, zašle server odpověď klientovi, že danou cestu není oprávněn použít.

3.4 Logování událostí

Pro sledování správné funkce sytému je zapotřebí zaznamenávat akce, které systém na příkaz uživatelů vykonává. Zaznamenávání chodu systému bylo také vyžadováno funkčním požadavkem pro administrátora FA2. Logy systému jsem rozdělil na dvě podmnožiny. Jedná se o:

- log přístupů na server
- log příkazů na změnu v databázi

V prvním případě vypisujeme přístupy na server jak do konzole, tak do souboru. Přesněji do souboru Server/logs/access.log. K tomuto účelu využívám knihovny morgan. V souboru je záznam uložen ve standardním formátu logu serveru Apache.

Pro záznam příkazů na změnu dat v databázi jsem vytvořil vlastní logovací funkci. K nahlédnutí je v ukázce 3.6. Všechny zaznamenané akce se nachází v souboru Server/logs/action.log.

```
1 function (action, user, resource, resourceId) {
2     stamp = new Date();
3     fs.appendFile(__dirname + "/../logs/actionLog",stamp.toString
          () + ' ' + action + " by user (id): " + user + " of
          resource " + resource + " with id: " + resourceId + '\n'
4               , "utf8", function (err) {
5                   if (err) {
6                       return console.error(err);
7                   }
8               });
9 };
```

Obrázek 3.6: Funkce zajišťující logování akcí

Vytvořené logy jsou dostupné jak lokálně z prostředí, ve kterém je spuštěna serverová aplikace, tak přes webovou stránku interní části systému, ke které má přístup uživatel, který je přihlášen jako administrátor.

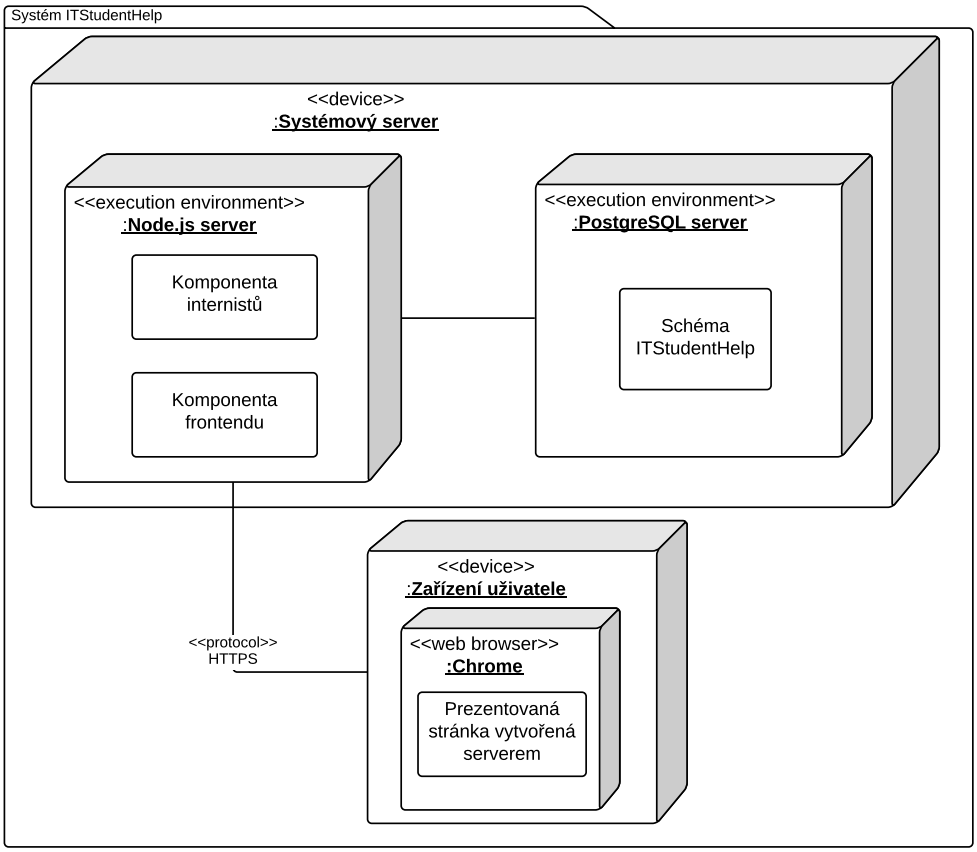
3.5 Diagram nasazení

Systém ještě není nasazen, avšak považuji za důležité uvést diagram nasazení 3.7 k pochopení součinnosti jednotlivých částí systému. Odevzdaná práce je připravena k nahrání na server, přes který bude možno řádně aplikaci otestovat, avšak reálnému nasazení stále brání několik skutečností.

Je třeba vyřešit zabezpečení komunikace mezi frontend a backend částmi vytvořené Igorem Kuľkou a Janem Kabelou. V backendové části je zřejmý pokus o využití tokenu k autentizaci požadavků, avšak ve frontendové části se s žádným autorizačním tokenem nepracuje.

Dále je nutné opravit popisy ve frontendové části, jelikož některé texty nejsou z hlediska české gramatiky správně.

V neposlední řadě v backendové části není vyřešena ekonomická strana projektu. Všechny úkoly jsou ohodnoceny fixním finančním obnosem. Pro vyřešení problému bych navrhl přidání sloupce do tabulky Úkol pro vedení ceny, za jakou byla zakázka vykonána. Toto řešení by poskytlo potřebnou flexibilitu pro změnu sazeb v budoucnosti.



Obrázek 3.7: Diagram nasazení systému

3.6 Adresářová struktura projektu

V této sekci popíši základní adresářovou strukturu konečného projektu. Následuje kostra systému s krátkým popisem jednotlivých adresářů a souborů. Nutno dodat, že zde nejsou rozepsány všechny souboru, které projekt obsahuje.

- `accounting` ..... adresář pro soubory generované externí účetní službě
- `cert` ..... adresář s certifikáty pro využití protokolu HTTPS
- `config`
  - `adminRoutes.js` ... nastavení cest dostupných pouze administrátorovi
  - `database.js` ..... nastavení parametrů pro přístup k databázi
  - `passport.js` ..... nastavení knihovny Passport
- `frontend` ..... frontend část systému vytvořená Igorem Kuľkou
- `functions` ..... obsahuje funkce využívané interní částí
  - `isLoggedIn.js` ..... kontroluje, zdali je uživatel přihlášen
  - `logger.js` ..... zaznamenává akce do logu

- `parseFormData.js` ..... parsuje požadavky z frontend části
- `images` ..... obsahuje obrázkové soubory
- `logs` ..... obsahuje logy systému
  - `access.log` ..... log přístupů na server
  - `action.log` ..... log požadavků na databázi
- `routes` ..... obsahuje funkční logiku cest serveru a interní části
  - `Accounts` ..... funkce uživatelských účtů
  - `Gadgets` ..... funkce Jana Kabely pro zasílání mailů atp.
  - `intern` ..... mnou vytvořená interní část
  - `Login` ..... funkce pro přihlášení uživatelů frontendu
  - `Password` ..... funkce pro změnu hesla
  - `Profile` ..... funkce zajišťující zisk informací o profilech uživatelů pro frontend
  - `Specializations` ..... funkce zajišťující zisk specializací pro frontend
  - `Tasks` ..... funkce zajišťující zisk informací o úkolech pro frontend
  - `faq.js` ..... funkce k získání dat pro sekci často kladené otázky
  - `routes.js` ..... definice cest backendové části Jana Kabely
- `static` ..... pro statická data
  - `homepage` ... obsahuje statická data pro domovskou stránku frontendu
- `views` ..... pohledy (view v MVC architektuře)
  - `intern` ..... pohledy části pro internisty
- `app.js` ..... spustitelná serverová aplikace
- `package.json` konfigurační soubor pro npm, obsahuje závislosti knihoven projektu

Hlavním úkolem této bakalářské práce bylo vytvoření části pro internisty. Největší část implementace interní části se nachází v adresáři routes/intern.

Zde nalezneme samotný model a řadič (controller) interní části. Modelové funkce se nacházejí v modelFunctions. Funkce řadiče jsou v souborech nesoucí název cest serveru, ke kterým se vztahují. Cesty implementované interní částí jsou následující:

- intern/ - hlavní stránka interní části
- intern/changePassword - změní heslo uživatele
- intern/login - přihlásí uživatele
- intern/login/forgottenPassword - odešle nové heslo na uživatelský e-mail
- intern/logout - odhlásí uživatele
- intern/customer/list - zobrazí seznam zákazníků
- intern/customer/list/tableData - vyplní tabulku pro zobrazení zákazníků
- intern/customer/edit - úprava údajů zákazníka
- intern/customer/detail - zobrazí veškeré údaje o zákazníkovi
- intern/customer/create - vytvoří zákazníka
- intern/employee/list - zobrazí seznam zaměstnanců

- intern/employee/list/tableData - vyplní tabulku pro zobrazení zaměstnance
- intern/employee/edit - upraví údaje zaměstnance
- intern/employee/detail - zobrazí veškeré údaje o zaměstnanci
- intern/employee/create - vytvoří zaměstnance
- intern/employee/changeRole - změní roli zaměstnance
- intern/faq - zobrazí často kladené otázky
- intern/faq/question - přidá otázku do často kladených otázek
- intern/faq/question/delete - odebere otázku z často kladených otázek
- intern/faq/answer - přidá odpověď k otázce v často kladených otázkách
- intern/faq/answer/delete - odebere odpověď od otázky v často kladených otázkách
- intern/logs - zobrazí nabídku logů systému
- intern/logs/actionLog - zobrazí log akcí
- intern/task/create - vytvoří zakázku
- intern/task/list - zobrazí aktivní zakázky v systému
- intern/task/list/tableData - vyplní tabulku pro zobrazení zakázek
- intern/task/archive - zobrazí hotové zakázky
- intern/task/archive/tableData - vyplní tabulku pro zobrazení hotových zakázek
- intern/task/validate - zobrazí seznam zakázek k validaci
- intern/task/validate/tableData - vyplní tabulku pro zobrazení zakázek k validaci
- intern/task/validate/detail - zobrazí veškeré údaje o zakázce k validaci
- intern/task/detail - zobrazí veškeré údaje o zakázce
- intern/task/edit - upraví údaje o zakázce
- intern/task/removeLector - odebere specialistu od zakázky
- intern/task/addLector - přidá specialistu k zakázce
- intern/task/done - označí zakázku jako splněnou
- intern/task/delete - vymaže zakázku

3.7 Testování

Testování interní části probíhalo dle scénářů, které jsou v příloze bakalářské práce. Scénáře demonstrují splnění všech funkčních požadavků uvedených v sekci 2.2.1.

Dále jsem se podílel na testování frontendové části. Byl jsem moderátorem v uživatelském testování zmíněné části. Frontend však v té době ještě nebyl integrován se serverovou aplikací.

Před nasazením do reálného provozu je nutno řádně otestovat systém jako celek nahráním na funkční server. Díky tomu bude možné otestovat systém při zátěži větším počtem uživatelů.

Závěr

V mé bakalářské práci se povedlo všechny části systému vytvořené ostatními studenty FIT ČVUT integrovat do jednoho kompaktního celku.

Výsledný systém obsahuje analýzu, návrh a implementaci backend části speciálně vytvořenou pro telefonistu a administrátora. Tato část poskytuje správu zakázek a uživatelů systému. Dále pak dovoluje zobrazení záznamů o fungování serverové aplikace.

Pro implementaci byla zvolena technologie splňující požadavky v samotném zadání bakalářské práce. Implementační platforma poskytuje možnost škálování při reálném nasazení.

V průběhu projektu jsem měl problémy s psaním serverové aplikace, jelikož samotná práce na backendové části systému nebyla explicitně definována. Z tohoto důvodu jsem několikrát narazil na případ, kdy jsme s Janem Kabelou pracovali na stejné funkčnosti, také díky tomu, že technologie Node.js byla pro oba novinkou.

Nakonec se však podařilo implementovat požadovanou funkčnost a díky projektu jsem získal znalost nových technologií jako Node.js a PostgreSQL.

Zásluhou tvorby grafického rozhraní pro interní uživatele systému jsem se také setkal se stylizací HTML formulářů.

Literatura

[1] Marketingtribune: *Studentaanhuis.nl geeft computerstudenten cadeau*. Naposledy navštíveno 12. 3. 2016, přeložil Jos Kannig. Dostupné z: `http://www.marketingtribune.nl/online/nieuws/2014/05/studentaanhuis.nl-geeft-computerstudenten-cadeau/index.xml`

[2] Neustad, I.; Arlow, J.: *UML 2 a unifikovaný proces vývoje aplikací*. Computer press a.s., 2004.

[3] Object Management Group: *Business Process Model and Notation*. Naposledy navštíveno 22. 6. 2016. Dostupné z: `http://www.bpmn.org/`

[4] Mozilla: *MVC architecture*. Naposledy navštíveno 1. 1. 2017, přeložil Štefan Töltési. Dostupné z: `https://developer.mozilla.org/en-US/Apps/Fundamentals/Modern_web_app_architecture/MVC_architecture`

[5] Node.js Foundation: *Node.js*. Naposledy navštíveno 27. 6. 2016, přeložil Štefan Töltési. Dostupné z: `https://nodejs.org/en/`

[6] Strongloop: *Why Node.js?* Naposledy navštíveno 27. 6. 2016, přeložil Štefan Töltési. Dostupné z: `https://strongloop.com/node-js/why-node/`

[7] Node.js Foundation: *Express*. Naposledy navštíveno 1. 1. 2017, přeložil Štefan Töltési. Dostupné z: `http://expressjs.com/`

[8] Jared Hanson: *Passport*. Naposledy navštíveno 6. 1. 2017, přeložil Štefan Töltési. Dostupné z: `http://passportjs.org/`

[9] SpryMedia Ltd: *DataTables*. Naposledy navštíveno 6. 1. 2017, přeložil Štefan Töltési. Dostupné z: `https://datatables.net/`

[10] W3C: *urlencoded*. Naposledy navštíveno 6. 1. 2017, přeložil Štefan Töltési. Dostupné z: `https://www.w3.org/TR/html401/interact/forms.html`

[11] Felixge: *formidable*. Naposledy navštíveno 5. 1. 2017. Dostupné z: `https://www.npmjs.com/package/formidable`

[12] itbiz: *API*. Naposledy navštíveno 5. 1. 2017. Dostupné z: `http://www.itbiz.cz/slovnik/informacni-technologie-it/api`

[13] *IT Slovník*. Naposledy navštíveno 21. 6. 2016. Dostupné z: `http://it-slovnik.cz/`

Příloha **A**

Seznam použitých zkratek a výrazů

API *„API je zkratka anglických slov application programming interface, což znamená rozhraní pro programování aplikací. Tento termín používá softwarové inženýrství v programování. Jde o sbírku procedur, funkcí či tříd nějaké knihovny (ale třeba i jiného programu nebo jádra operačního systému), které může využívat programátor, který knihovnu využívá. API určuje, jakým způsobem se funkce knihovny mají volat ze zdrojového kódu programu“* [12]

Autentizace *„Proces k ověření identity. Slouží k ochraně dat.“* [13]

Autorizace *„Proces při němž dochází k ověření údajů při vstupu do nějakého systému. Při tomto procesu se vyhodnocuje, zda jsou zadané přístupové údaje korektní a zda má uživatel právo ke vstupu.“* [13]

backend část systému ke které nemá běžný uživatel přístup

BPMN Business Process Model and Notation - norma pro znázornění podnikových procesů

Callback funkce spustitelná funkce, která se volá po dodání dat asynchronní funkcí

cookies malé soubory uložené prohlížečem sloužící k uchování stavu klienta

E-R diagram Entity-relationship diagram, znázorňuje vztah entit databáze

framework software sloužící jako rámec při programování sofwarových projektů.

frontend část systému přístupná běžnému uživateli.

HTML HyperText Markup Language - značkovací jazyk pro tvorbu webových stránek

HTTP HyperText Transfer Protocol - protokol využívaný k zasílání HTML dokumentů

JSON Formát pro textovou reprezentaci objektů jazyku JavaScript.

session relace HTTP protokolu, session umožňuje uložení informací o uživateli na serveru

SQL Structured Query Language - standardizovaný dotazovací jazyk pro relační databáze.

URL Uniform Resource Locator - řetězec znaků sloužící ke specifikaci umístění zdrojů na internetu.

Příloha B

Obsah přiloženého CD

- `API` ..... obsahuje definici API pro frontend a backend
  - `itstudenthelp.apib` samotná definice API vytvořená Igorem Kuľkou
- `readme.txt` ..... stručný popis obsahu CD
- `schema` ..... schéma databáze vygenerované pomocí SchemaSpy
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `Database` ..... soubory pro tvorbu databáze
    - `Server` ..... zdrojové kódy serverové aplikace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `testing` ..... scénáře testování
- `thesis.pdf` ..... text práce ve formátu PDF